Embedded USB Host CDC-ACM Class Driver User's Guide

Version 2.10

For use with USBH CDC-ACM Class Driver Versions 2.05 and above

Date: 27-Aug-2014 15:23

Table of Contents

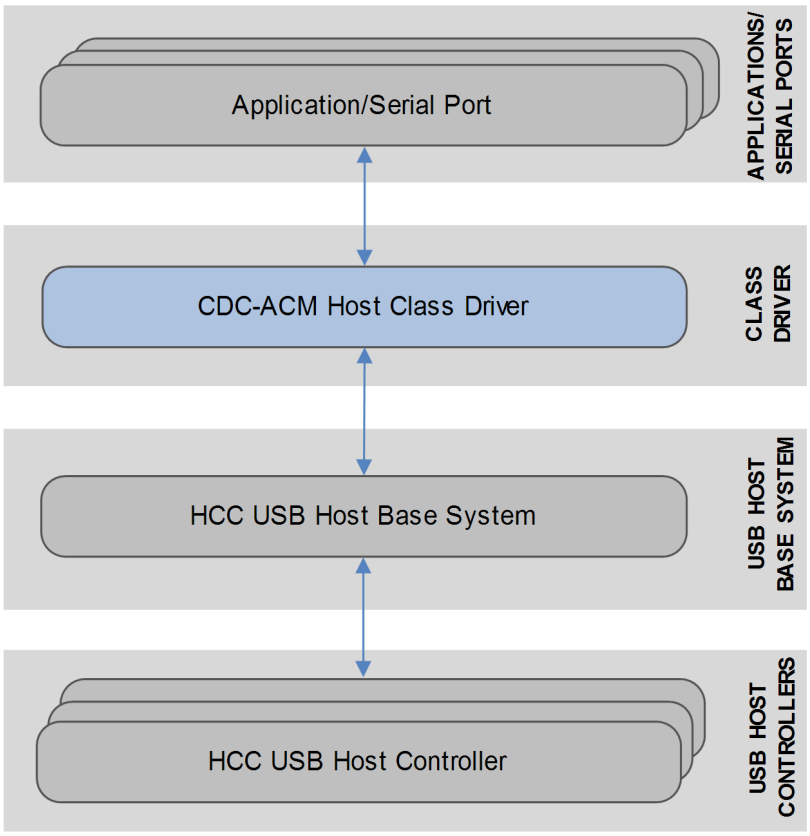
1 System Overview

1.1 Introduction

This guide is for those who want to implement an Embedded USB CDC-ACM host class driver to control Control Device Class - Abstract Control Module (CDC-ACM) USB devices. The CDC-ACM class provides a serial interface for connecting devices such as modems to an embedded system.

The **usbd_cd_cdc_acm** package provides a CDC-ACM host class driver for a USB stack. The system allows a USB serial port device to be plugged into the host and recognized as a remote serial port. The package provides a set of interface functions for controlling access to the serial port.

The system structure is shown in the diagram below:



The lower layer interface is designed to use HCC Embedded's USB Host Interface Layer. This layer is standard over different host controller implementations; this means that the code is the same, whichever HCC USB host controller it is interfaced to. For detailed information about this layer, refer to the *HCC USB Host Base System User's Guide* that is shipped with the base system.

The package provides a set of API functions for controlling access to a device. These are described here, with separate sections for module and line management.

1.2 Feature Check

The main features of the class driver are the following:

- It conforms to the HCC Advanced Embedded Framework.
- It is compatible with all HCC USB host controllers.
- It supports all devices that conform to the USB CDC-ACM specification.
- It supports multiple devices connected simultaneously.
- It supports line state and control management.
- It uses a system of callbacks for user-specified events.

1.3 Packages and Documents

Packages

This table lists the packages that you need in order to use this module:

Package	Description
hcc_base_doc	This contains the two guides that will help you get started.
usbh_base	The USB host base package. This is the framework used by USB class drivers to communicate over USB using a specific USB host controller package.
usbh_cd_cdc_acm	The USB device CDC-ACM host class driver package described by this document.

Documents

Readers should note the points in the HCC Documentation Guidelines on the HCC documentation website.

HCC Firmware Quick Start Guide

This document describes how to install packages provided by HCC in the target development environment. Also follow the *Quick Start Guide* when HCC provides package updates.

HCC Source Tree Guide

This document describes the HCC source tree. It gives an overview of the system to make clear the logic behind its organization.

HCC USB Host Base System User's Guide

This document defines the USB host base system upon which the complete USB stack is built.

HCC Embedded USB Host CDC-ACM Class Driver User's Guide

This is this document.

2 Source File List

This section describes all the source code files included in the system. These files follow the HCC Embedded standard source tree system, described in the *HCC Source Tree Guide*. All references to file pathnames refer to locations within this standard source tree, not within the package you initially receive.

> **Note:** Do not modify any files except the configuration file.

2.1 API Header File

The file **src/api/api_usbh_cdcacm.h** is the only file that should be included by an application using this module. For details of the API functions, see API.

2.2 Configuration File

The file **src/config/config_usbh_cdcacm.h** contains all the configurable parameters of the system. Configure these as required. For details of these options, see Configuration Options.

2.3 Source Code

The file **src/usb-host/class-drivers/cdc-acm/usbh_cdcacm.c** is the main code for the module. **This file should only be modified by HCC**.

2.4 Version File

The file **src/version/ver_usbh_cdcacm.h** contains the version number of this module. This version number is checked by all modules that use this module to ensure system consistency over upgrades.

3 Configuration Options

Set the system configuration options in the file **src/config/config_usbh_cdcacm.h**. This section lists the available configuration options and their default values.

3.1 Standard Options

USBH_CDCACM_MAX_UNITS

The maximum number of CDC-ACM serial units the system can handle. This includes the maximum units per interface and the number of connected devices. The default is 1.

> **Note:** If you configure two devices, these take the unit IDs UID 0 and UID 1. (These are referred to by the parameter uid in the API functions.)

USBH_CDCACM_MAX_UNITS_PER_INTERFACE

The maximum number of data units the system can handle on one interface. The default is 1.

USBH_CDCACM_RXBUF_COUNT

The number of buffers to use for receiving in the background. The default is 2.

When a device is plugged in reception automatically starts. When data arrives the user gets a notification/polls for data. In the meantime reception to the next buffer can be started to keep the performance as high as possible.

USBH_CDCACM_RXBUF_SIZE

The size of a receive buffer. The default is 512. This must be a multiple of 64 for full speed systems, and a multiple of 512 for high speed systems.

USBH_CDCACM_COM_INTERFACE_ENABLE

This enables the COM interface. The default is 1.

USBH_CDCACM_COMBUF_SIZE

Size of the COM interface buffer. Retaining 64, the default, is a safe option.

3.2 Using Alternates

If support for non-standard CDC-ACM devices is required, you can define alternate class, subclass and protocol numbers. The system allows COM channel handling for both on the same data interface.

> **Note:** Some devices implement the COM INT endpoint on the same interface where data endpoints are present; this is not standard CDC-ACM but it is supported by this class driver.

USBH_CDCACM_ALT0_DATA_CLASS

The alternate 0 data class. The default is 0xff.

USBH_CDCACM_ALT0_DATA_SCLASS

The alternate 0 subclass. The default is 0x00.

USBH_CDCACM_ALT0_DATA_PROTOCOL

The alternate 0 data protocol. The default is 0x00.

USBH_CDCACM_ALT1_DATA_CLASS

The alternate 1 data class. The default is 0xff.

USBH_CDCACM_ALT1_DATA_SCLASS

The alternate 1 subclass. The default is 0xff.

USBH_CDCACM_ALT1_DATA_PROTOCOL

The alternate 1 data protocol. The default is 0xff.

4 API

This section documents the Application Programming Interface. It includes all the functions that are available to an application program.

4.1 Module Management Functions

usbh_cdcacm_init

Use this function to initialize the class driver and allocate the required resources.

> **Note:** You must call this function before any other function is used.

Format

```
int usbh_cdcacm_init ( void )
```

Arguments

Parameter
None.

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
Else	See Error Codes.

usbh_cdcacm_start

Use this function to start the class driver.

> **Note:** You must call **usbh_cdcacm_init()** before this function.

Format

```
int usbh_cdcacm_start ( void )
```

Arguments

Parameter
None.

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
Else	See Error Codes.

usbh_cdcacm_stop

Use this function to stop the class driver.

Format

```
int usbh_cdcacm_stop ( void )
```

Arguments

Parameter
None.

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
Else	See Error Codes.

usbh_cdcacm_delete

Use this function to delete the class driver and release the associated resources.

Format

```
int usbh_cdcacm_delete ( void )
```

Arguments

Parameter
None.

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
Else	See Error Codes.

4.2 Line Management Functions

usbh_cdcacm_send

Use this function to send data over the CDC-ACM channel.

If TX notification is registered, this function returns immediately. You can call the function **usbh_cdcacm_get_send_state()**, either before or after the notification, to obtain the status of the transfer.

Format

```
int usbh_cdcacm_send (
    t_usbh_unit_id   uid,
    uint8_t *        buf,
    uint32_t         length )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
buf	A pointer to the buffer to send.	uint8_t *
length	The number of bytes to send.	uint32_t

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
Else	See Error Codes.

usbh_cdcacm_get_send_state

Use this function to get the completion code of the last send.

If TX notification is registered, a call to **usbh_cdcacm_send()** returns immediately. After the notification (or before) you can call **usbh_cdcacm_get_send_state()** to obtain the status of the transfer.

Format

```
int usbh_cdcacm_get_send_state ( t_usbh_unit_id uid )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id

Return Values

Return value	Description
Completion code of the last send.	Successful execution.
Else	See Error Codes.

usbh_cdcacm_receive

Use this function to receive data on the CDC-ACM channel.

Format

```
int usbh_cdcacm_receive (
    t_usbh_unit_id   uid,
    uint8_t *        buf,
    uint32_t         max_length,
    uint32_t *       rlength )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
buf	A pointer to the buffer to receive the data.	uint8_t *
max_length	The size of the buffer.	uint32_t
rlength	On return, the number of bytes written to the buffer.	uint32_t *

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
Else	See Error Codes.

usbh_cdcacm_get_receive_state

Use this function to get the state of the last receive.

- If RX notification is registered, it is sufficient to call **usbh_cdcacm_receive()** after the notification.
- If there is no notification, you can use this function to find whether data is available.

Format

```
int usbh_cdcacm_get_receive_state ( t_usbh_unit_id uid )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id

Return Values

Return value	Description
Completion code of the last receive.	Successful execution.
Else	See Error Codes.

usbh_cdcacm_rx_chars

Use this function to get the number of characters in the receive buffer.

> **Note:** If the host sends a zero length packet to the device, this function returns with zero bytes read, even if CDC_WAIT_FOREVER is specified.

Format

```
uint32_t usbh_cdcacm_rx_chars ( t_usbh_unit_id uid )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id

Return Values

Return value	
Number of characters in the receive buffer.	Successful execution.
Else	See Error Codes.

usbh_cdcacm_get_line_coding

Use this function to get the current configuration of the specified line.

Format

```
int usbh_cdcacm_get_line_coding (
    t_usbh_unit_id   uid,
    uint32_t *       br,
    uint8_t *        b,
    uint8_t *        p,
    uint8_t *        s )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
br	The baud rate.	uint32_t *
b	Bits.	uint8_t *
p	Parity.	uint8_t *
s	Stop bits.	uint8_t *

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
Else	See Error Codes.

usbh_cdcacm_set_line_coding

Use this function to configure the specified serial line.

Format

```
int usbh_cdcacm_set_line_coding (
    t_usbh_unit_id   uid,
    uint32_t         br,
    uint8_t          b,
    uint8_t          p,
    uint8_t          s )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
br	The baud rate.	uint32_t
b	Bits.	uint8_t
p	Parity.	uint8_t
s	Stop bits.	uint8_t

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
Else	See Error Codes.

usbh_cdcacm_get_serial_state

Use this function to get the serial state. This is sent through the communication interface.

> **Note:** If USBH_CDCACM_COM_INTERFACE_ENABLE is not set, this function is not available.

Format

```
uint16_t usbh_cdcacm_get_serial_state ( t_usbh_unit_id uid )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id

Return Values

Return value	Description
A bitmask of the serial state.	Successful execution.
Else	See Error Codes.

usbh_cdcacm_set_control_line_state

Use this function to configure the control line.

Format

```
int usbh_cdcacm_set_control_line_state (
    t_usbh_unit_id   uid,
    uint8_t          rts,
    uint8_t          dtr )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
rts	The RTS state.	uint8_t
dtr	The DTR state.	uint8_t

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
Else	See Error Codes.

usbh_cdcacm_get_port_hdl

Use this function to get the port handle.

Format

```
t_usbh_port_hdl usbh_cdcacm_get_port_hdl ( t_usbh_unit_id uid )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id

Return Values

Return value	Description
Port handle.	Successful execution.
USBH_PORT_HDL_INVALID	Invalid port handle.

usbh_cdcacm_present

Use this function to check whether a device is connected.

Format

```
int usbh_cdcacm_present ( t_usbh_unit_id uid )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id

Return Values

Return value	Description
Zero.	No device is present.
Non-zero.	A device is present.

usbh_cdcacm_register_ntf

Use this function to register a notification function for a specified event type.

When a device is connected or disconnected, or one of the specific events for this type of device occurs, the notification function is called.

> **Note:** It is the user's responsibility to provide any notification functions required by the application. Providing such functions is optional.

Format

```
int usbh_cdcacm_register_ntf (
    t_usbh_unit_id   uid,
    t_usbh_ntf       ntf,
    t_usbh_ntf_fn    ntf_fn )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
ntf	The notification type.	t_usbh_ntf
ntf_fn	The notification function to be used when an event occurs.	t_usbh_ntf_fn

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
Else	See Error Codes.

4.3 Error Codes

If a function executes successfully, it returns with a USBH_SUCCESS code, a value of zero. The following table shows the meaning of the error codes:

Return Code	Value	Description
USBH_SUCCESS	0	Successful execution.
USBH_SHORT_PACKET	1	IN transfer completed with short packet.
USBH_PENDING	2	Transfer still pending.
USBH_ERR_BUSY	3	Another transfer in progress.
USBH_ERR_DIR	4	Transfer direction error.
USBH_ERR_TIMEOUT	5	Transfer timed out.
USBH_ERR_TRANSFER	6	Transfer failed to complete.
USBH_ERR_TRANSFER_FULL	7	Cannot process more transfers.
USBH_ERR_SUSPENDED	8	Host controller is suspended.
USBH_ERR_HC_HALTED	9	Host controller is halted.
USBH_ERR_REMOVED	10	Transfer finished due to device removal.
USBH_ERR_PERIODIC_LIST	11	Periodic list error.
USBH_ERR_RESET_REQUEST	12	Reset request during enumeration.
USBH_ERR_RESOURCE	13	OS resource error.
USBH_ERR_INVALID	14	Invalid identifier/type (HC, EP HDL, and so on).
USBH_ERR_NOT_AVAILABLE	15	Item not available.
USBH_ERR_INVALID_SIZE	16	Invalid size.
USBH_ERR_NOT_ALLOWED	17	Operation not allowed.
USBH_ERROR	18	General error.

4.4 Types and Definitions

t_usbh_ntf_fn

The **t_usbh_ntf_fn** definition specifies the format of the notification function. It is defined in the USB host base system in the file **api_usb_host.h**.

Format

```
int  ( * t_usbh_ntf_fn )(
    t_usbh_unit_id   uid,
    t_usbh_ntf       ntf )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
ntf	The notification code.	t_usbh_ntf

Notification Codes

The standard notification codes shown below are defined in the USB host base system in the file **api_usb_host.h**.

Notification	Value	Description
USBH_NTF_CONNECT	1	Connection notification code.
USBH_NTF_DISCONNECT	2	Disconnection notification code.

The additional notification codes provided by this module are as follows:

Notification	Value	Description
USBH_NTF_CDCACM_RX	USBH_NTF_CD_BASE + 0	Data has been received.
USBH_NTF_CDCACM_TX	USBH_NTF_CD_BASE + 1	Data has been sent.
USBH_NTF_CDCACM_NTF	USBH_NTF_CD_BASE + 2	The line state has been changed.

Bit Number Codes

This table shows the bit number codes:

Code	Default Setting
USBH_CDCSER_BITS_5	5
USBH_CDCSER_BITS_6	6
USBH_CDCSER_BITS_7	7
USBH_CDCSER_BITS_8	8
USBH_CDCSER_BITS_16	16

Parity Codes

This table shows the parity codes:

Code	Default Setting
USBH_CDCSER_PARITY_NONE	0
USBH_CDCSER_PARITY_ODD	1
USBH_CDCSER_PARITY_EVEN	2
USBH_CDCSER_PARITY_MARK	3
USBH_CDCSER_PARITY_SPACE	4

Stop Bit Codes

This table shows the stop bit codes:

Code	Default Setting
USBH_CDCSER_STOP_1	0
USBH_CDCSER_STOP_1_5	1
USBH_CDCSER_STOP_2	2

LST Codes

This table shows the LST codes:

Code	Default Setting
USBH_CDCSER_LST_OE	1u << 6
USBH_CDCSER_LST_PE	1u << 5
USBH_CDCSER_LST_FE	1u << 4
USBH_CDCSER_LST_RINGS	1u << 3
USBH_CDCSER_LST_BREAK	1u << 2
USBH_CDCSER_LST_DSR	1u << 1
USBH_CDCSER_LST_DCD	1u << 0

Note: These are unique bits in a set of bit values.

5 Integration

This section specifies the elements of this package that need porting, depending on the target environment.

5.1 OS Abstraction Layer (OAL)

All HCC modules use the OS Abstraction Layer (OAL) that allows the module to run seamlessly with a wide variety of RTOSes, or without an RTOS.

The class driver uses the following OAL components:

OAL Resource	Number Required
Tasks	0
Mutexes	1
Events	0

5.2 PSP Porting

The Platform Support Package (PSP) is designed to hold all platform-specific functionality, either because it relies on specific features of a target system, or because this provides the most efficient or flexible solution for the developer.

The module makes use of the following standard PSP functions:

Function	Package	Component	Description
psp_memcpy()	psp_base	psp_string	Copies a block of memory. The result is a binary copy of the data.

The module makes use of the following standard PSP macros:

Macro	Package	Component	Description
PSP_RD_LE16	psp_base	psp_endianness	Reads a 16 bit value stored as little-endian from a memory location.
PSP_RD_LE32	psp_base	psp_endianness	Reads a 32 bit value stored as little-endian from a memory location.
PSP_WR_LE16	psp_base	psp_endianness	Writes a 16 bit value to be stored as little-endian to a memory location.
PSP_WR_LE32	psp_base	psp_endianness	Writes a 32 bit value to be stored as little-endian to a memory location.